# ESE5320: System-on-a-Chip Architecture

Day 6: September 20, 2023
Data-Level Parallelism

Preclass





## Today

Data-level Parallelism

- For Parallel Decomposition (Part 1)
- Architectures (Part 2)
- Concepts (Part 3)
- NEON







## Message

- Data Parallelism easy basis for decomposition
  - When things are data parallel, exploiting for parallelism relatively easy
- Data Parallel architectures can be compact
  - pack more computations onto a fixed-size Integrated Circuit chip
  - *OR* perform computation in less area







## Preclass 1

- 500 news articles
  - 3000 words each
- Count total occurrences of a string
- How can we exploit data-level parallelism on task?
- How much parallelism can we exploit?
  - If don't exploit parallelism within article?
  - If do exploit parallelism within article?







## Parallel Decomposition







## Data Parallel

- Data-level parallelism can serve as an organizing principle for parallel task decomposition
- Run computation on independent data in parallel

## Exploit

- Can exploit with
  - Threads
  - Pipeline Parallelism
  - Instruction-level Parallelism
  - Fine-grained Data-Level Parallelism





## Performance Benefit

- Ideally linear in number of processors (resources)
- Resource Bound:
  - $T_{dp} = (T_{single} \times N_{data}) / P$
- $T_{single}$ = Latency on single data item
- $T_{dp} = T_{single} \times \lceil N_{data} / P \rceil$





## Vector

- In Math – 1D array of numbers
- Vector A = [2, 3, 5, 7]
- Vector B = [4, 6, 8, 9]
- A+B = [6, 9, 13,16]





## Preclass 2 Common Examples

- What are common examples of DLP?
  - Numerical Linear Algebra
    - E.g. Vector Add, Matrix-Vector Multiplication
  - Simulation
  - Signal or Image Processing
  - Optimization





## Hardware Architectures

Part 2





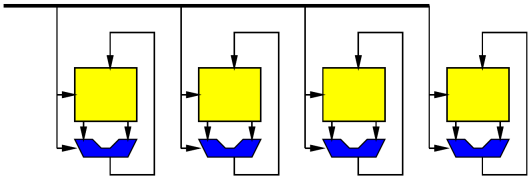
## Idea

- If we're going to perform the same operations on different data, exploit that to reduce area, energy
- Reduced area means can have more computation on a fixed-size IC chip.

# SIMD

- Single Instruction Multiple Data

**Shared Instruction**

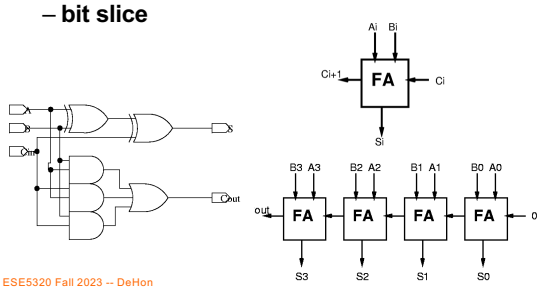




13

# Ripple Carry Addition

- Can define logic for each bit, then assemble:
  - **bit slice**

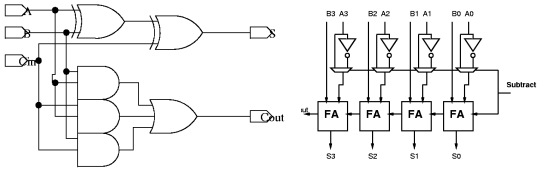





14

# Arithmetic Logic Unit (ALU)

- Observe:
  - with small tweaks can get many functions with basic adder components

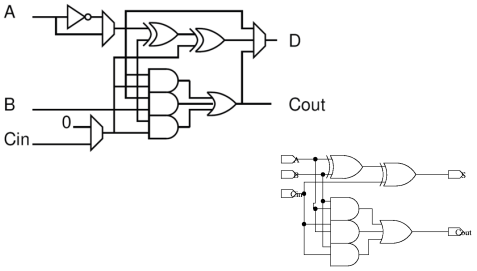





15

# Arithmetic and Logic Unit

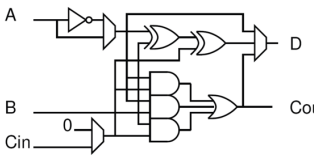
ALU






16

# ALU Functions




- A+B w/ Carry
- B-A
- A xor B (squash carry)
- A*B (squash carry)
- /A

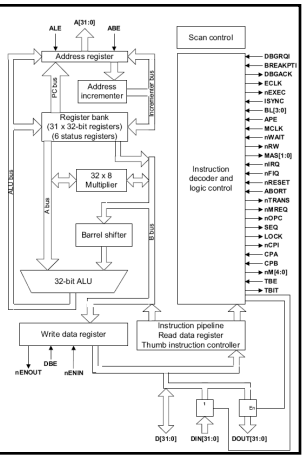
**Key observation:** every ALU bit does the same thing on different bits of the data word(s).



17

# ARM v7 Core

- ALU is Key compute operator in a processor

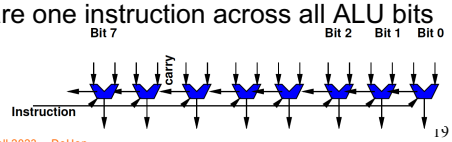





18

## W-bit ALU as SIMD

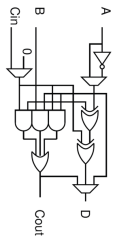
- Familiar idea
- A W-bit ALU (W=8, 16, 32, 64, …) is SIMD
- Each bit of ALU works on separate bits
  - Performing the same operation on it
    - Trivial to see bitwise AND, OR, XOR
    - Also true for ADD (each bit performing Full Adder)
- Share one instruction across all ALU bits

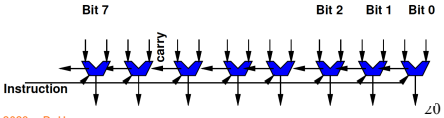





19

## ALU Bit Slice






20

## Preclass 4

- What do we get when add 65280 to 257
  - 32b unsigned add?
  - 16b unsigned add?



21

## ALU vs. SIMD ?

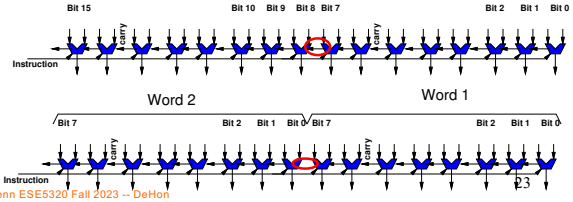
- What's different between
  - 128b wide ALU
  - SIMD datapath supporting eight 16b ALU operations



22

## ALU vs. SIMD Example

- Concretely show:
  - 16b wide ALU
  - SIMD datapath with two 8b wide ALUs






23

## ALU vs. SIMD ?

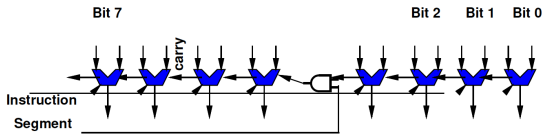
- How could we get both operations from the **same** hardware?
  - 128b wide ALU
  - SIMD datapath supporting eight 16b ALU operations



24

## Segmented Datapath

- Add a few gates to convert a wide datapath into one supporting a set of smaller operations
  - Just need to squash the carry at points






25

## Segmented Datapath

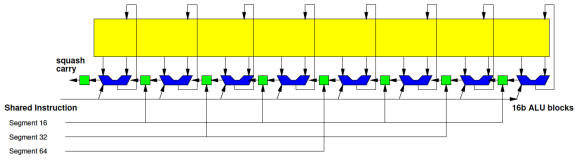
- Add a few gates to convert a wide datapath into one supporting a set of smaller operations
  - Just need to squash the carry at points
- But need to keep instructions (description) small
  - So typically have limited, homogeneous widths supported



26

## Segmented 128b Datapath

- 1x128b, 2x64b, 4x32b, 8x16b




(only 4 cases, so can decode from 2b)



27

## Vector

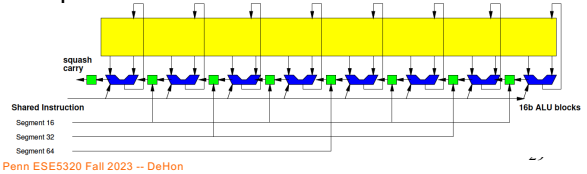
- In Math – 1D array of numbers
- Here – 1D array of numbers that we operate upon with SIMD operations
  - Perform the same operations on
- Vector A = [2, 3, 5, 7]
- Vector B = [4, 6, 8, 9]
- A+B = [6, 9, 13,16]



28

## Terminology: Vector Lane

- Each of the separate segments called a **Vector Lane**
  - Length of vector hardware natively supports
- For 16b data on 128b datapath, this provides 8 vector lanes

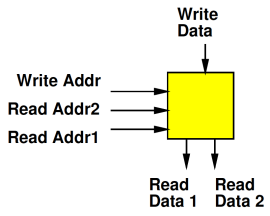





29

## Register File

- Small Memory
- Usually with multiple ports
  - Ability to perform multiple reads and writes simultaneously
- Small
  - To make it fast (small memories fast)
  - Multiple ports are expensive

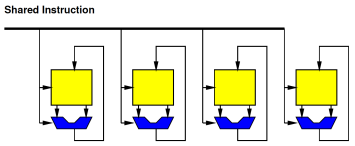




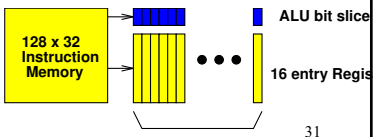

30

# SIMD Datapath

- Logical View:




- Layout View:






31

# Preclass 5

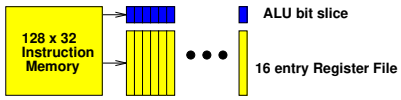
- Look at Area of datapath
- Imagine filling an IC die with copies (instances) of datapath
  - If datapath smaller, can fit more of them into fixed die area
- Look at computation this provides



32

# Preclass 5

| W | A(W) | % datapath | Instances | Peak 16b ops |
|---|---|---|---|---|
| 16 | | | | |
| 64 | | | | |
| 128 | | | | |
| 256 | | | | |



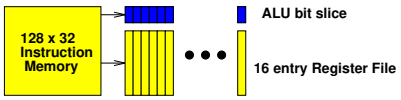



33

# Preclass 5

| W | A(W) | % datapath | Instances | Peak 16b ops | Ratio |
|---|---|---|---|---|---|
| 16 | 1.1 | 9 | 91 | 91 | 1.0 |
| 64 | | | | | |
| 128 | 1.8 | 43 | 56 | 448 | 4.9 |
| 256 | | | | | |

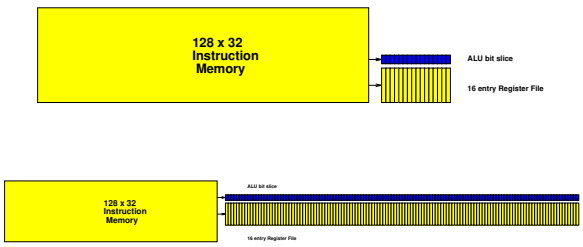
Complete: 64, 256






34

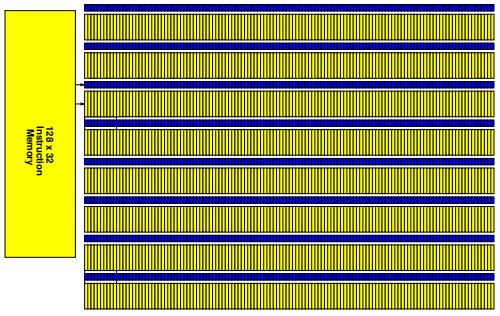
# To Scale Comparison






35

# To Scale W=1024






36

## Take Away

- Wider SIMD → greater potential density
  - More compute per cm$^2$ of silicon
- There is a point of diminishing return
  - Once over half the area in datapath
- Wider SIMD may be inefficient if do not have enough data-level parallelism

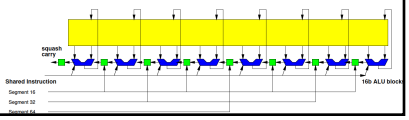




## Preclass 6: Vector Length

- May not match physical hardware length
  - Logical (application need): Vector Length
  - Physical (hardware provide): Vector Lanes

```
vadd(int *a, int *b, int *res,
     int vector_length) {
    for (int i=0;i<vector_length;i++)
    res[i]=a[i]+b[i];
    }
```








## Preclass 6: Vector Length

- May not match physical hardware length
  - Logical (application need): Vector Length
  - Physical (hardware provide): Vector Lanes
- Efficiency (utilization) when
  - Vector length < Vector Lanes? (5, 8)
  - Vector length > Vector Lanes? (24, 8)
  - Vector length % (Vector Lanes) !=0 (13, 8)
    - E.g. vector length 13, for 8 vector lanes

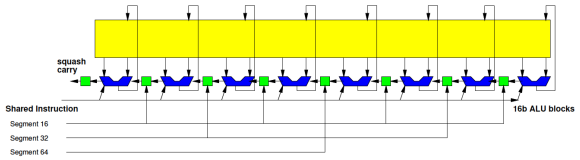




## Performance

- Resource Bound

  $T_{vector} = \lceil N_{op}/VL \rceil \times T_{cycle}$








## Preclass 3: Opportunity

- Don't need 64b variables for lots of things
- Natural data sizes?
  - Audio samples?
  - Input from A/D?
  - Video Pixels?
  - X, Y coordinates for 4K x 4K image?





## Vector Computation

- Easy to map to SIMD flow if can express computation as operation on vectors
  - Vector Add
  - Vector Multiply
  - Dot Product

# Concepts

Part 3



43

## Terminology: Scalar

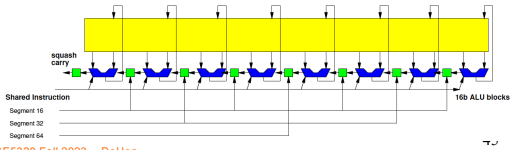
- Simple: non-vector
- When we have a vector unit controlled by a normal (non-vector) processor core often need to distinguish:
  - Vector operations that are performed on the vector unit
  - Normal=non-vector=scalar operations performed on the base processor core



44

## Vector Register File

- Need to be able to feed the SIMD compute units
  - Not be bottlenecked on data movement to the SIMD ALU
- Wide RF (register file) to supply
- With wide path to memory

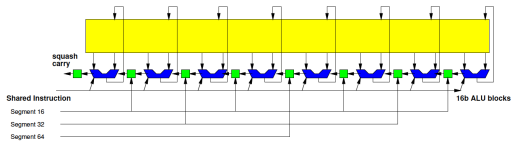





45

## Point-wise Vector Operations

- Easy – just like wide-word operations (now with segmentation)

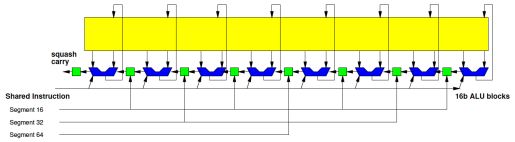





46

## Point-wise Vector Operations

- …but alignment matters.
- If not aligned, need to perform data movement operations to get aligned






47

## Ideal

- for (i=0;i<64;i=i++)
  - c[i]=a[i]+b[i]
- No data dependencies
- Access every element
- Number of operations is a multiple of number of vector lanes



48

## Skipping Elements?

- How does this work with datapath?
  - Assume loaded a[0], a[1], …a[63] and b[0], b[1], …b[63] into vector register file
- for (i=0;i<64;i=i+2)
  - c[i/2]=a[i]+b[i]





## Stride

- Stride: the distance between vector elements used
- for (i=0;i<64;i=i+2)
  - c[i/2]=a[i]+b[i]
- Accessing data with stride=2





## Load/Store

- Strided load/stores
  - Some architectures will provide strided memory access that compact when read into register file
- Scatter/gather
  - Some architectures will provide memory operations to grab data from different places to construct a dense vector





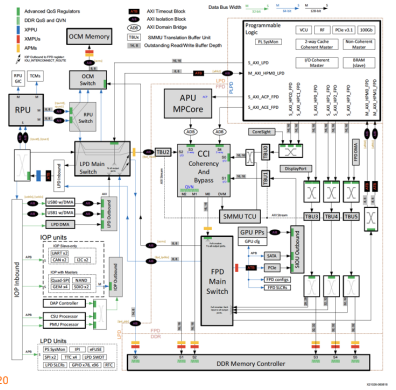
## Neon

ARM Vector Accelerator





## Programmable SoC

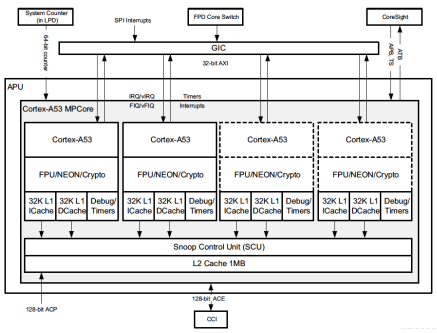



UG1085 Xilinx UltraScale Zynq TRM (p27)





## APU MPcore




UG1085 Xilinx UltraScale Zynq TRM (p53)

## Neon Vector

- 128b wide register file, 16 registers
- Support
  - 2x64b
  - 4x32b (also Single-Precision Float)
  - 8x16b
  - 16x8b



55

## Sample Instructions

- VADD – basic vector
- VCEQ – compare equal
  - Sets to all 0s or 1s, useful for masking
- VMIN – avoid using if's
- VMLA – accumulating multiply
- VPADAL – maybe useful for reduce
  - Vector pair-wise add
- VEXT – for "shifting" vector alignment
- VLDn – deinterleaving load



56

## ARM Cortex A53 (Ultra96) (similar to A-7 Pipeline)

2-issue
In-order
1 NEON pipe
128b wide
8-stage pipe

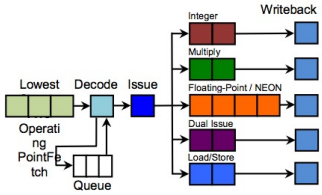



Figure 1 Cortex-A7 Pipeline

https://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/3

https://arstechnica.com/gadgets/2011/10/arms-new-cortex-a7-is-tailor-made-for-android-superphones/



57

## Big Ideas

- Data Parallelism easy basis for decomposition
- Data Parallel architectures can be compact – pack more computations onto a chip
  - SIMD, Pipelined
  - Benefit by sharing (instructions)
  - Performance can be brittle
    - Drop from peak as mismatch



58

## Admin

- Remember feedback
- Reading for Day 7 online
- HW3 due Friday
- HW4 out
  - Including partner assignments
    - Board holders – contact partner soon



59